

Ejercicio Sorting

December 13, 2021

1 Inciso A

IDEA

Tengo las T tandas ordenadas. Tomo el primer elemento de cada tanda y armo una tripla $(\text{nombre}, \text{velocidad}, \text{listaDeLaQueFueSacado})$ que voy a poner en un arreglo (de tamaño t).

Luego, sobre ese arreglo hago heapify con costo $O(t)$. La comparación que utilizo para armar el minheap va a ser por el elemento "velocidad" de la tripla. Ante empates, tomo cualquiera (por ejemplo, el de la izquierda). Ahora tengo un minheap donde están los 1 ros elementos de las T tandas.

For i in $0 \dots p$:

- Saco el elemento min del heap
- Creo una tupla $(\text{nombre}, \text{velocidad})$ con los elementos del min
- Tomo el valor $\text{listaDeLaQueFueSacado}$ del min
- Obtengo el sig de $\text{listaDeLaQueFueSacado}$ (si es que tiene)
- Creo la tupla nueva $(\text{nombre}, \text{velocidad}, \text{listaDeLaQueFueSacado})$
- La meto en el heap

Cuando ya saque los p elementos, terminé.

ALGORITMO

```
primeros(in p:nat, in tandas: lista(lista(participante)))
  -> res: lista(participante) {

  res <- listaVacía()           // 0(1)

  unoPorTanda <- CrearArreglo(tam(tandas)) // 0(t)

  itTandas <- crearIterador(tandas)      // 0(1)

  indice <- 0
  while (haySiguiente(itTandas)) {      // total: 0(t)
    tanda <- Siguiente(itTandas) // referencia a la lista 0(1))
    if (no vacía(tanda)) {           // 0(1)
      particip <- Primero(tanda) // 0(1)
      tanda <- Fin(tanda)           // Fin es 0(1)
      unoPorTanda[indice] <- < particip.id, particip.tiempo, itTandas > // 0(1)
      indice <- indice + 1           // 0(1)
    }
    Avanzar(itTandas)                // 0(1)
  }

  heapConParticipantes <- heapify(unoPorTanda) // minHeap en 0(t)

  for i in 0..p {                    total: 0(p * log t)
    infoParticipante <- dameMin(heapConParticipantes) // 0(log t)

    // recuerdo: infoParticipante tiene <id, tiempo, iterador>
    p <- CrearParticipante(infoParticipante.id, infoParticipante.tiempo) // 0(1)
    res.agregarAlFinal(p)           // 0(1)

    // Ahora actualizo el heap
    tanda <- Siguiente(infoParticipante.iterador) // referencia a la lista 0(1)

    if (no vacía(tanda)) {
      nuevoParticip <- Primero(tanda) // 0(1)
      tanda <- Fin(tanda)             // 0(1)

      // mantengo el iterador a la posición de la tanda
      // (que se mantiene por más que cambien los participantes que contiene)
      infoNuevoParticip <- CrearTupla(nuevoParticip.id,
                                      nuevoParticip.tiempo, infoParticipante.iterador) // 0(1)

      agregar(heapConParticipantes, infoNuevoParticipante) // 0(log t)
    }
  }
```

```
    }
}
```

Para el heap, usamos la siguiente función de comparación:

```
compararTuplas(in t1: tupla, t2: tupla) -> res: tupla {
  if(t2[1] < t1[1]){
    res <- t2
  } else {
    res <- t1
  }
}
```

COMPLEJIDAD Y CORRECTITUD

El algoritmo se divide en tres partes:

- Crear el array con los T primeros, que tiene un costo de $O(t)$ de crear el array y sacar el 1ro de cada tanda porque tengo acceso directo (es decir, saco el 1ro de t tandas)
- Hacer el heapify, que tiene costo $O(t)$ como vimos en la teorica
- Sacar p veces el min elem del heap (costo $O(\log t)$) y meter el nuevo de la tanda que corresponde ($O(\log(t))$). Como se hace p veces, es $O(p \cdot \log(t))$

Como siempre saco el minimo del heap, y siempre tengo el siguiente valor a sacar de cada una de las tandas, al finalizar el algoritmo saque los primeros p de todas las tandas.

De este modo, si sumamos los costos, nuestro algoritmo tiene una complejidad de $O(t + t + p \cdot \log(t)) = O(t + p \cdot \log(t))$

2 Inciso B

Para poder generar dos listas con p profesionales y p amateurs, primero debo dividir las listas: para cada tanda, genero una lista con sus profesionales y otra con sus amateurs.

De esta forma, luego corro el algoritmo original `primeros(p, tandasProfesionales)` y `primeros(p, tandasAmateurs)` y obtengo ambas listas.

El costo es el de recorrer todas las tandas ($O(n)$, con n participantes totales) y luego, el de utilizar 2 veces el algoritmo `primeros`.

Es decir: $O(n + 2 \cdot (t + p \cdot \log(t))) = O(n + t + p \cdot \log(t))$ que es, si tenemos en cuenta que $n \geq t$, $O(n + p \cdot \log(t))$